A Data Driven Approach to Designing Adaptive Trustworthy Systems

Ravishankar K. Iyer

(with A. Sharma, K. Pattabiraman, Z. Kalbarczyk, …

Center for Reliable and High-Performance Computing
Department of Electrical and Computer Engineering and Coordinated Science Laboratory
University of Illinois at Urbana-Champaign

http://www.crhc.uiuc.edu/DEPED

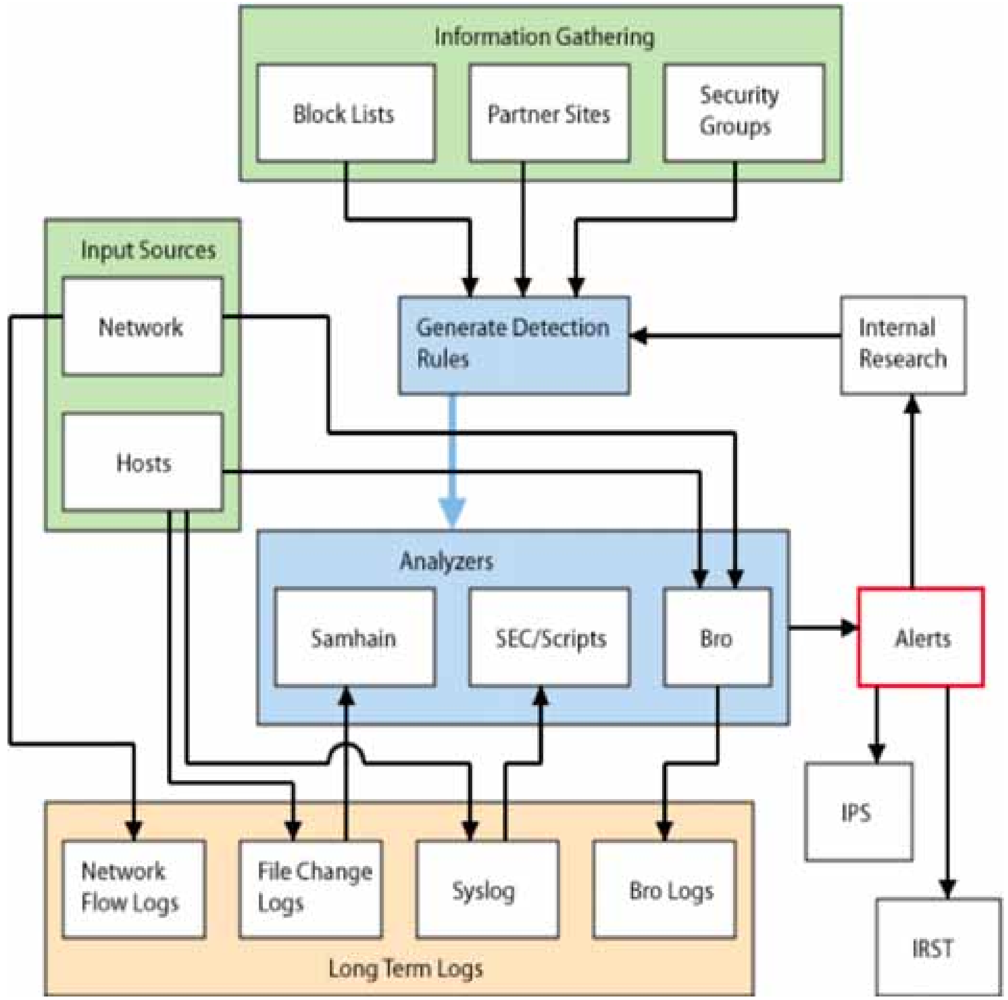
ILLINOIS

Analysis of Security Incidents at NCSA: a Large Open Networked Computing Environment

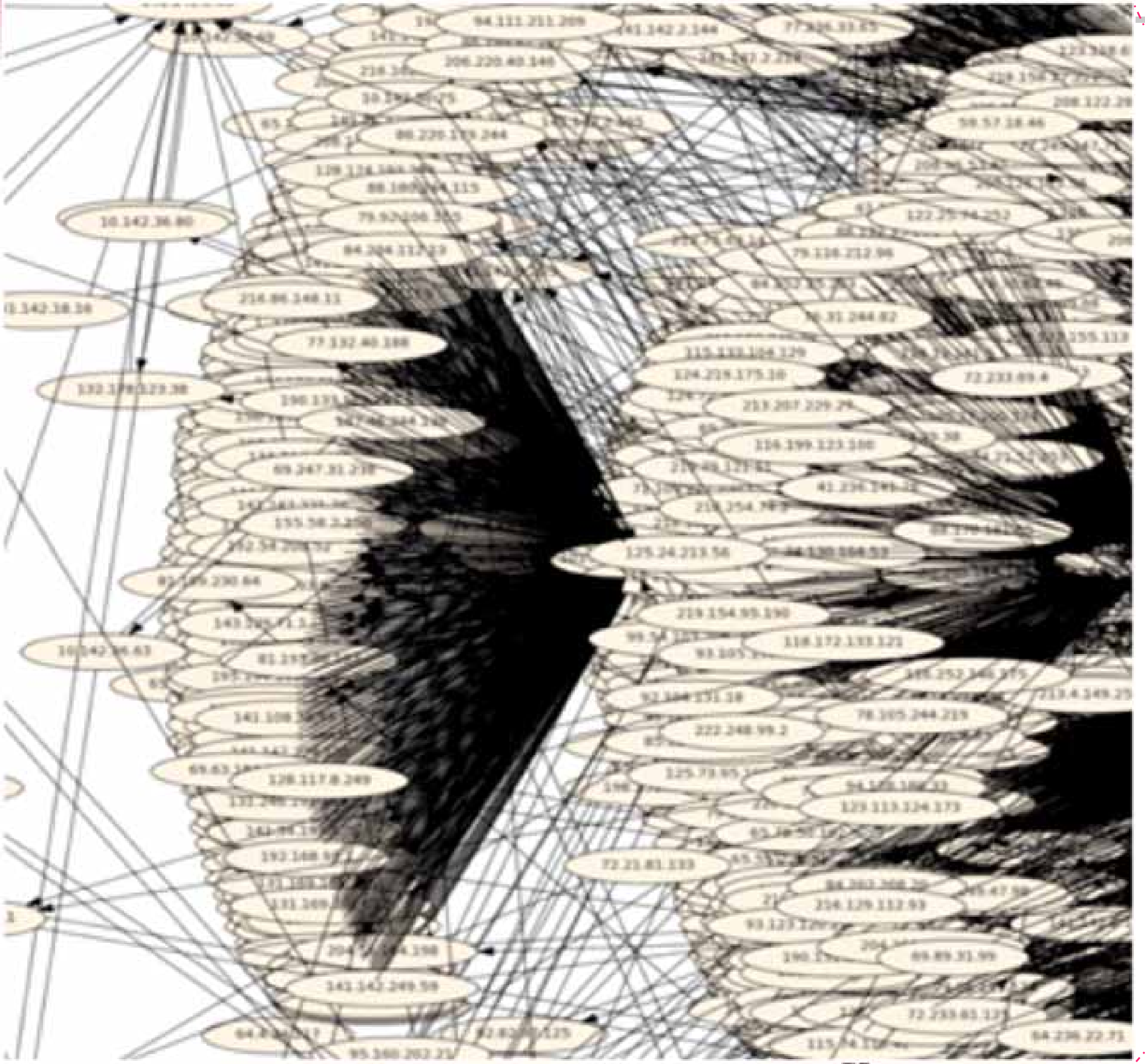
ILLINOIS

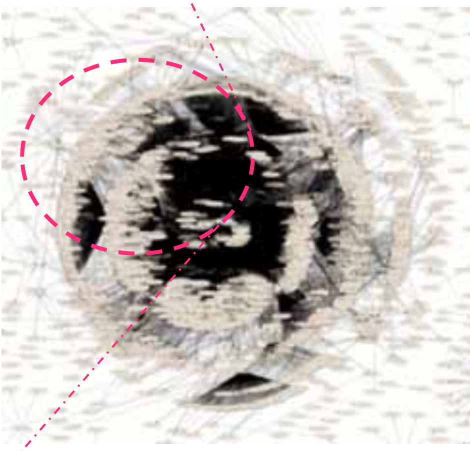
NCSA Target System

Number of hosts	5000+ (clusters, workstations, laptops)
Number of Active Users:	6000+
Network	Class B (/16)
Monitoring Links	10Gb pipes
Monitoring Tools	- IDS (4.5GB daily logs) - Network Flow (2.0G) - File integrity check - Central Syslog (1.5G)
OS Types	Linux, AIX, Solaris, OS-X, Windows



Five-Minute Snapshot of In-and-Out Traffic within NCSA





ILLINOIS

Incident Data
(124 real Incidents + 26 investigations)

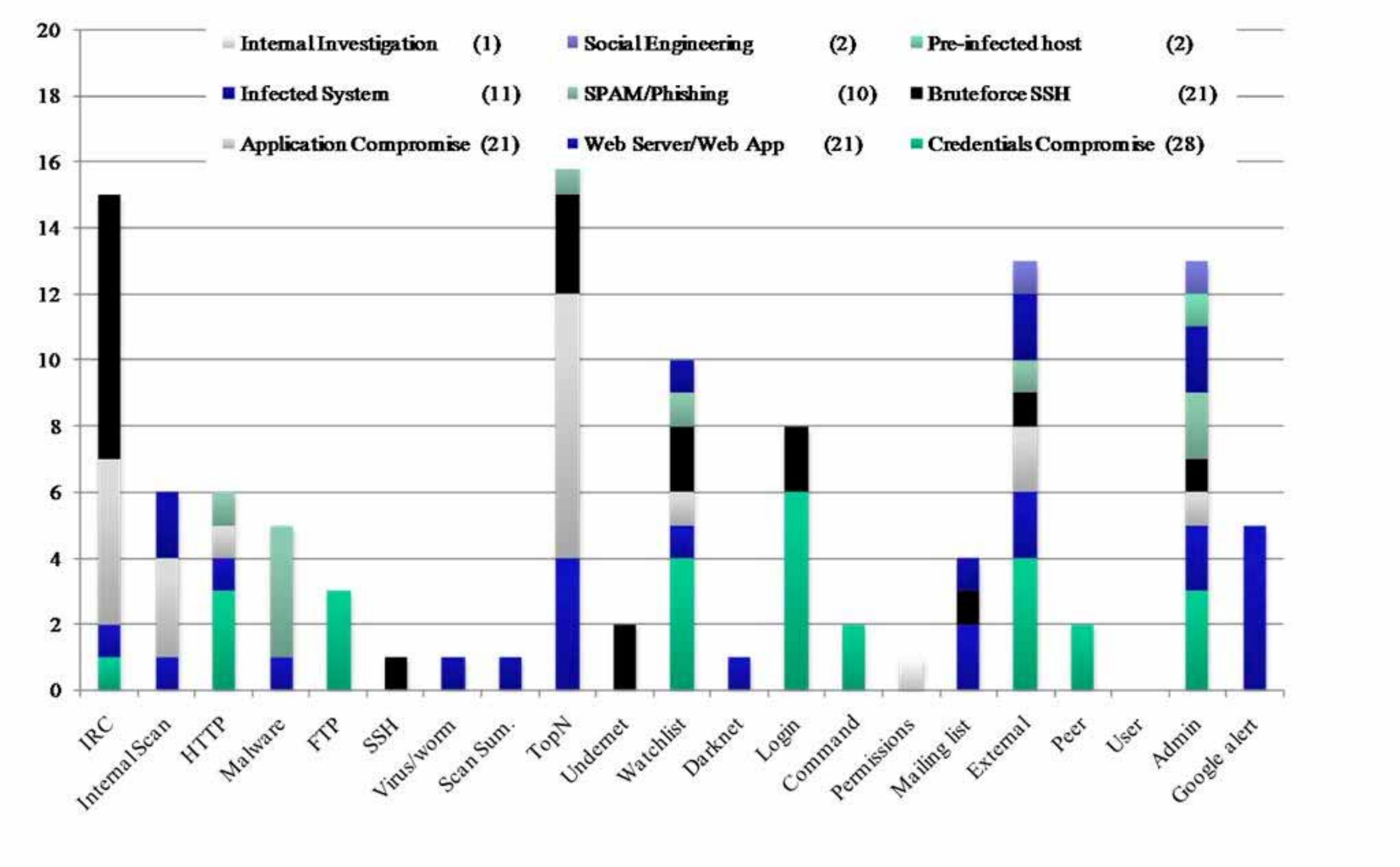
Sample records

ID	Incident Type	Monitor / Alert	Exploit used	Misuse	Privilege obtained	Attack Phase	Severity
1	Application Compromise	Flows/ TopN	xp_cmdshell MSSQL Server	Warez unauthorized media	root	Attack relay/ misuse	Medium
2	Infected System	IDS/Blaster	W32.Welchi worm	Scan ICMP 2048	user	Attack relay/ misuse	Low
3	Credentials compromise	Syslog/ Profiling	Stolen credentials	Sniff credentials	root	Breach	High

Monitors and Alert distribution

#	IDS alerts Packet/Protocol Analyzers								Flows alerts Traffic Analyzers				Syslog Profile		FIM Host	Other	No alerts 3rd Party Notification				
	IRC	Internal Scan	HTTP	Malware	FTP	SSH	Virus/ worm	Scan	TopN	Undernet	Watchlist	Darknet	Login	Command	File change	Google alert	Mailing list	External	Peer	User	Admin
INC 124	15	6	6	5	3	1	1	1	**18**	2	11	1	10	4	1	5	4	14	2	0	14
INV 150	16	6	7	6	3	1	1	1	**27**	2	13	1	11	4	1	5	4	19	3	1	18

Distribution of Incident Types vs Alert Types



Major Observations: All Incidents

- The majority of incidents (55%) due to attacks on authentication mechanisms with varying levels of sophistication
  - password guessing (bruteforce SSH),
  - exploiting a vulnerability, e.g., VNC null session,
  - installing trojaned versions of SSH and SSHD to sniff passwords and target public-private key pairs (credential stealing)
- The same alert can be triggered by different attacks.
  - the basic steps followed by attacks (of same category) in penetrating the system are often similar regardless of the vulnerability exploited
- Anomaly-based detectors are seven times more likely to capture an incident than are signature-based detectors
  - signatures are specialized to detect the presence (or download) of a known malicious binary but can be easily subverted
  - the signature-based detectors have fewer false positives compared to the anomaly-based detectors.

ILLINOIS

Analysis of an Example Incident
(Credentials Stealing Category: Total 32 incidents)

- **An IDS alert shows suspicious download** on a production system (victim: *xx.yy.ww.zz*) using http protocol from remote host *aa.bb.cc.dd*.

May 16 03:32:36 %187538 start xx.yy.ww.zz:44619 > aa.bb.cc.dd:80
May 16 03:32:36 %187538 GET /.0/ptrat.c (200 "OK" [2286] server5.bad-host.com)

- The file is suspect because
  - This particular system is not expected to download any code apart from patches and system updates, and then only from authorized sources
  - The downloaded file is a C language source code
- The server the source was downloaded from not a formal software distribution repository.
- ***The alert does not reveal what caused the potentially illegal download request***

Correlations with Other Logs

- **Network flows reveal further connections with other hosts** in close time proximity to the occurrence of the download:
  - SSH connection from IP address 195.aa.bb.cc
  - Multiple FTP connections to ee.ff.gg.hh, pp.qq.rr.ss.

```
09-05-16 03:32:27 v tcp 195.aa.bb.cc.35213 -> xx.yy.ww.zz.22 80  96  8698 14159   FIN
09-05-16 03:33:36 v tcp xx.yy.ww.zz.44619 -> aa.bb.cc.dd.http 8  6  698 4159   FIN
09-05-16 03:34:37 v tcp xx.yy.ww.zz.53205 -> ee.ff.gg.hh.ftp 1699 2527 108920 359566 FIN
09-05-16 03:35:39 v tcp xx.yy.ww.zz.39837 -> pp.qq.rr.ss.ftp 236  364  15247  546947 FIN
```

- ***SSH connection record does not reveal***
  - ***Whether authentication was successful***
  - ***What credentials were used to authenticate the user***

Correlation with *syslog* Alerts

- *syslog* confirms a user login from *195.aa.bb.cc*, which is unusual, based on the know user profile and behavior patterns

---

May 16 03:32:27 host sshd[7419]: Accepted password for user from 195.aa.bb.cc port 35794 ssh2

- ***Four data points established from the anlysis***
  - ***A suspicious source code was downloaded,***
  - ***The user login occurred at nearly the same time as the download,***
  - ***First time login from IP address 195.aa.bb.cc,***
  - ***Additional communication on other ports (FTP)***

Additional (Manual) Analysis

- Search of all files owned or created by this user found a footprint left behind by a credential-stealing exploit.

-rwxrwxr-x 1 user user 3945 May 16 03:37 /tmp/libno_ex.so.1.0

- ***The additional analysis showed***
  - ***The library file libno_ex.so.1.0 is known to be created when an exploit code for a known vulnerability (cve-2009-1185) is successfully executed***
  - ***File is owned by the user whose account was stolen and used to login to the system***
  - ***The attacker obtained root privileges in the system and replaced the SSHD daemon with a trojaned version***
    - ***Harvesting more user credentials***

Observations: Incident Analysis

- No single available tool can perform this kind of analysis
- Need to correlate:
  - data from different monitors
  - system logs
  - human expertise
- Need to develop techniques to pre-empt an attacker actions
  - potentially let the attacker to progress under *probation* (or tight scrutiny) until the real intentions are clear

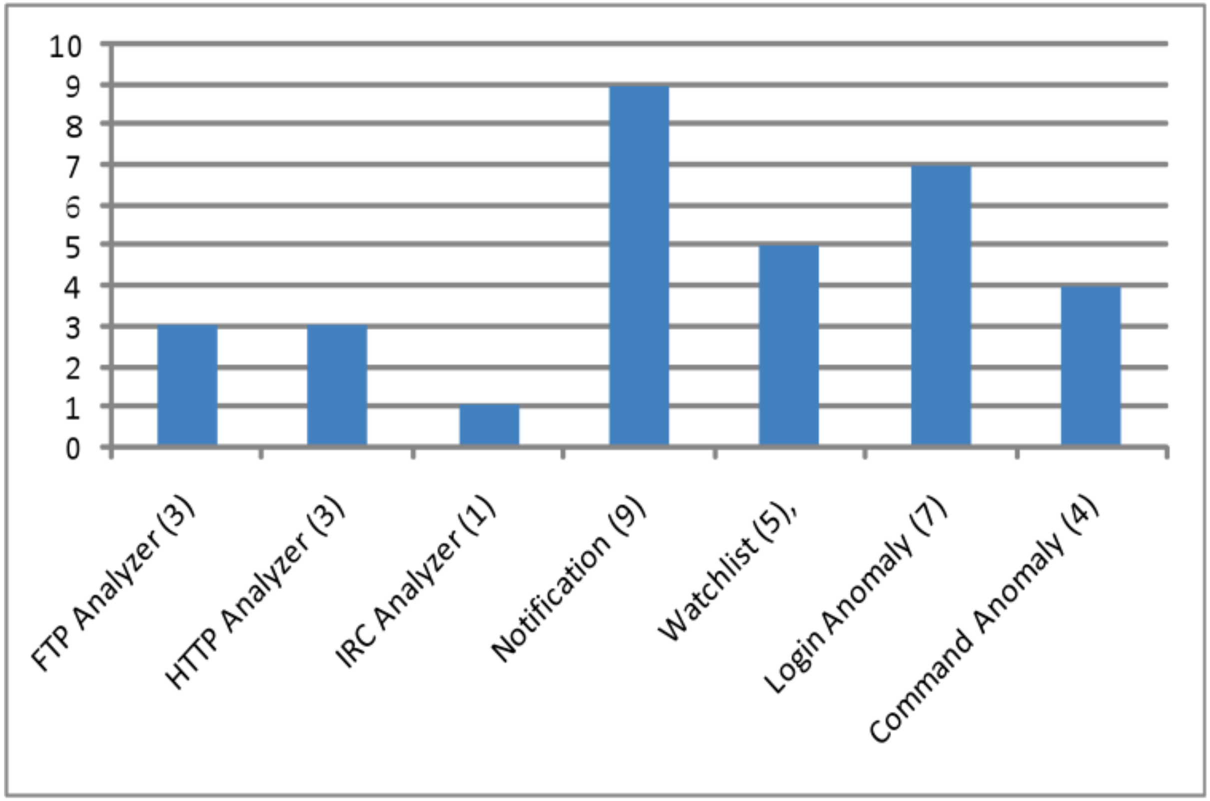
ILLINOIS

Credentials Stealing Attacks

- Initial investigation of security incidents indicated that nearly 26% (32/124) of the incidents analyzed involved credentials stealing
- 31 out of 32 incidents attackers came into the system with a valid credential of an NCSA user account
  - Attackers rely on their access to an external repository of valid credentials to harvest more credentials
  - Availability of valid credentials makes boundary protections (e.g., reliance only on a firewall) insufficient for this type of attacks.
  - More scrutiny in monitoring user actions is required

Detection of Credentials Stealing Incidents

About 28% (9/32) of credentials stealing incidents missed by the monitors, i.e., none ofiincident was discovered by external notification



IDS = 7 Incidents
Flows=5 Incidents
Syslogs = 11 incidents

ILLINOIS

Characteristics of Credentials Stealing Incidents

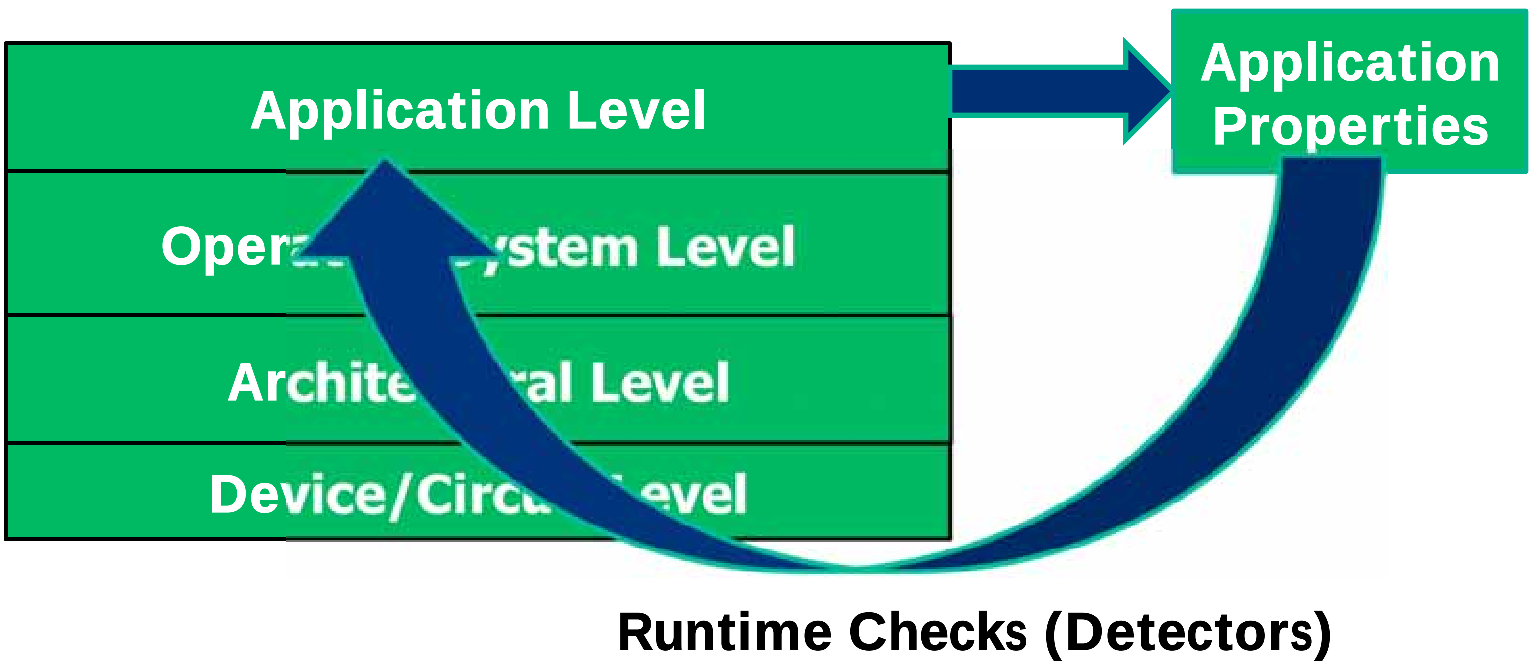
- Attackers obtained access to the host
  - Using a stolen password (78%)
  - A public key (16%)
  - Combination of multiple authentication means
  - (password + gssapi-with-mic or password + publickey) (6%).
- 31% of incidents miscreants obtain root on the compromised host and install a rootkit and/or sniffer by using a local root escalation exploit
- In 9% of incidents, the attacker downloaded additional tools to scan for a vulnerability in the NFS file system.
- Attackers came in with valid credentials (over 90%)! Insider like attacks

Application-aware Checking
TRUSTED ILLIAC Project

ILLINOIS

Application-aware Detection

- **App-aware: Use application properties to derive error and attack detectors (runtime checks)**
  - Achieve high-detection coverage with low overheads
  - Detect only attacks and errors that matter to the application
  - Ensure that attack and error is detected before propagation



Unified Design Framework

Reliability

Security

Identify critical variables and their location within a program

Apply heuristics, e.g., *fanouts*, to identify critical variables.

Use application semantics to identify security critical variables, e.g., a password

↓

Critical Variable Recomputation

Static program analysis: Extract backward slices of critical variables

Information-flow signatures (IFS)

Generate correctness checks for data values in critical program locations

Generate checks to verify that value is produced by legitimate instructions.

↓

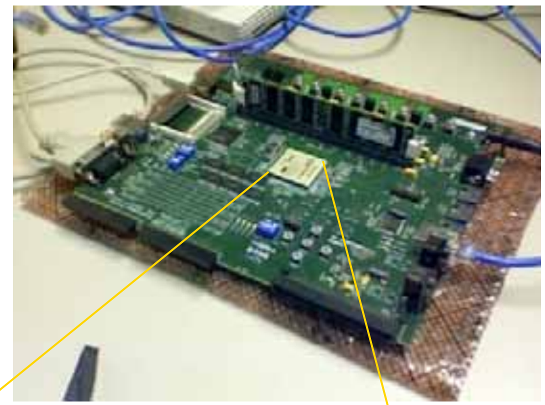
Runtime checking to ensure integrity of critical variables according to the slice

Techniques and Attack/Error Models

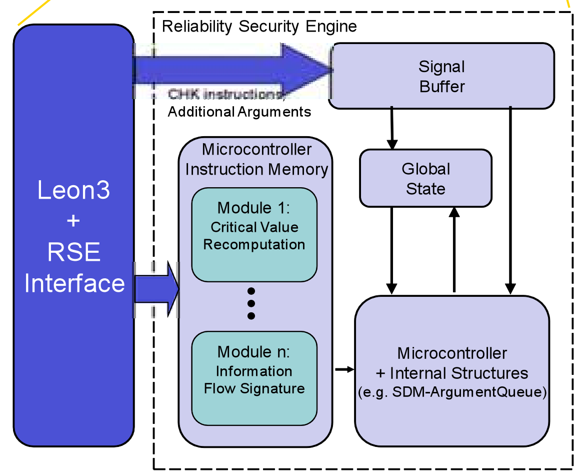
- **Selectively enforce source-level properties of writes to critical data at runtime**
- **Techniques:**
  - IFS (information flow signatures) – protects critical data integrity
  - CVR (critical value re-computation) – verifies correctness of critical data computation
- **Attack Models**
  - Memory corruption attacks
  - Control and/or data flow change
  - Insider attacks (malicious libraries, 3rd party plugins)
  - Binary modifications – illegal downloads
- **Fault Models**
  - Soft errors
  - Memory corruption errors
  - Race conditions and/or atomicity violations

Hybrid Implementation (hw + sw)

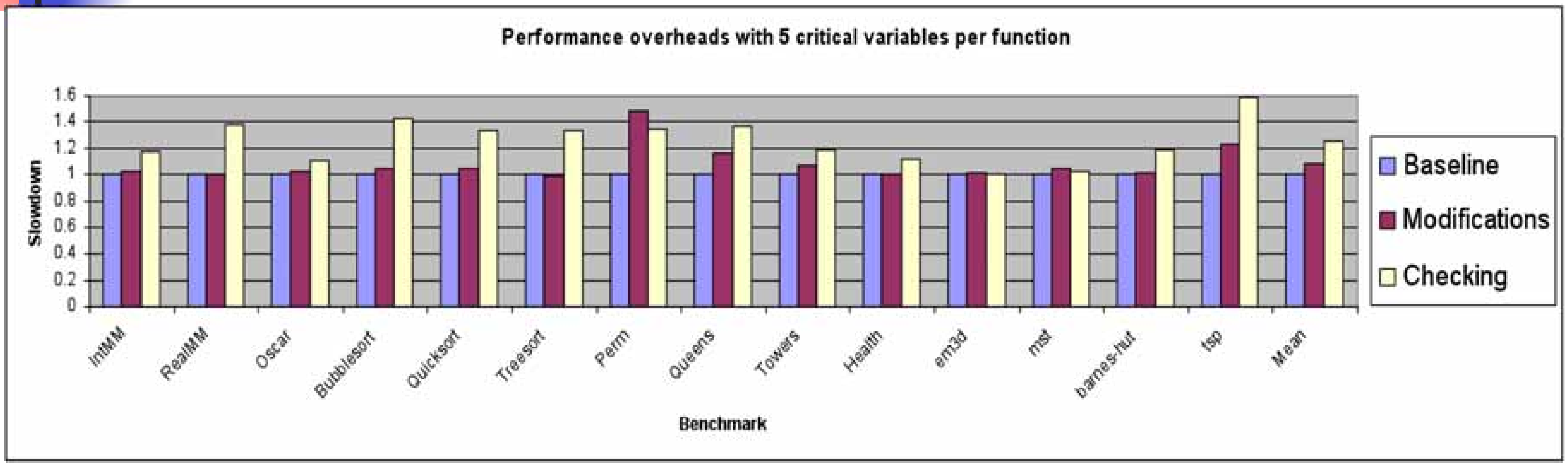
- Runtime enforcement using combination of hardware and software



- Single hardware framework host modules providing reliability and security protection
- FPGA-based prototype evaluated on embedded programs and network applications (e.g., OpenSSH)
- Performance overhead = 1% to 30 % (depending on the application)
- Area overhead = 4% to 20 % (relative to Leon3 processor)



Results (5 critical var per func)

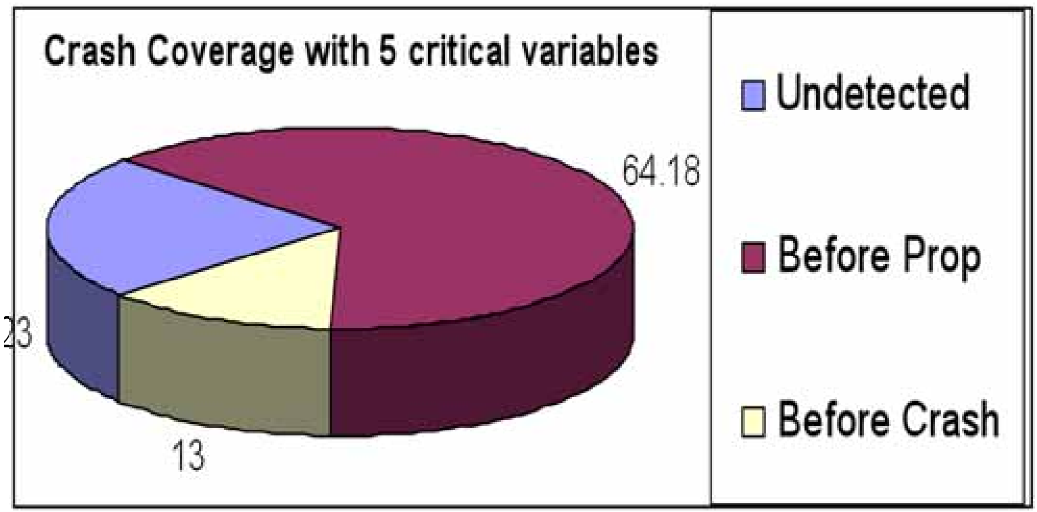


Average Perf. Overhead

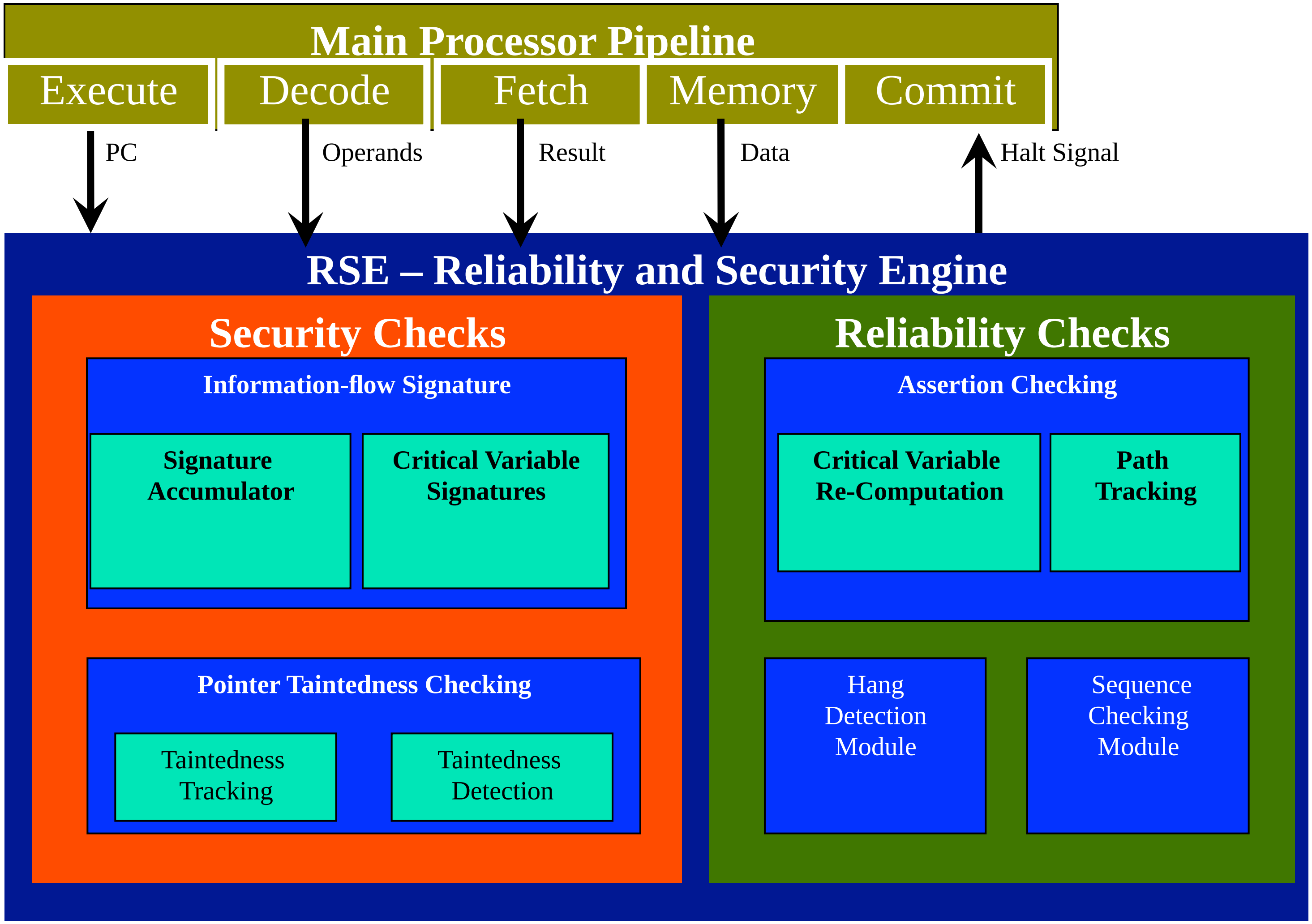
- Checking = 25%
- Modification = 8%
- **Total = 33 %**

Average Coverage

- Before Prop = 64 %
- Before Crash = 13%
- **Total Detected = 77 %**

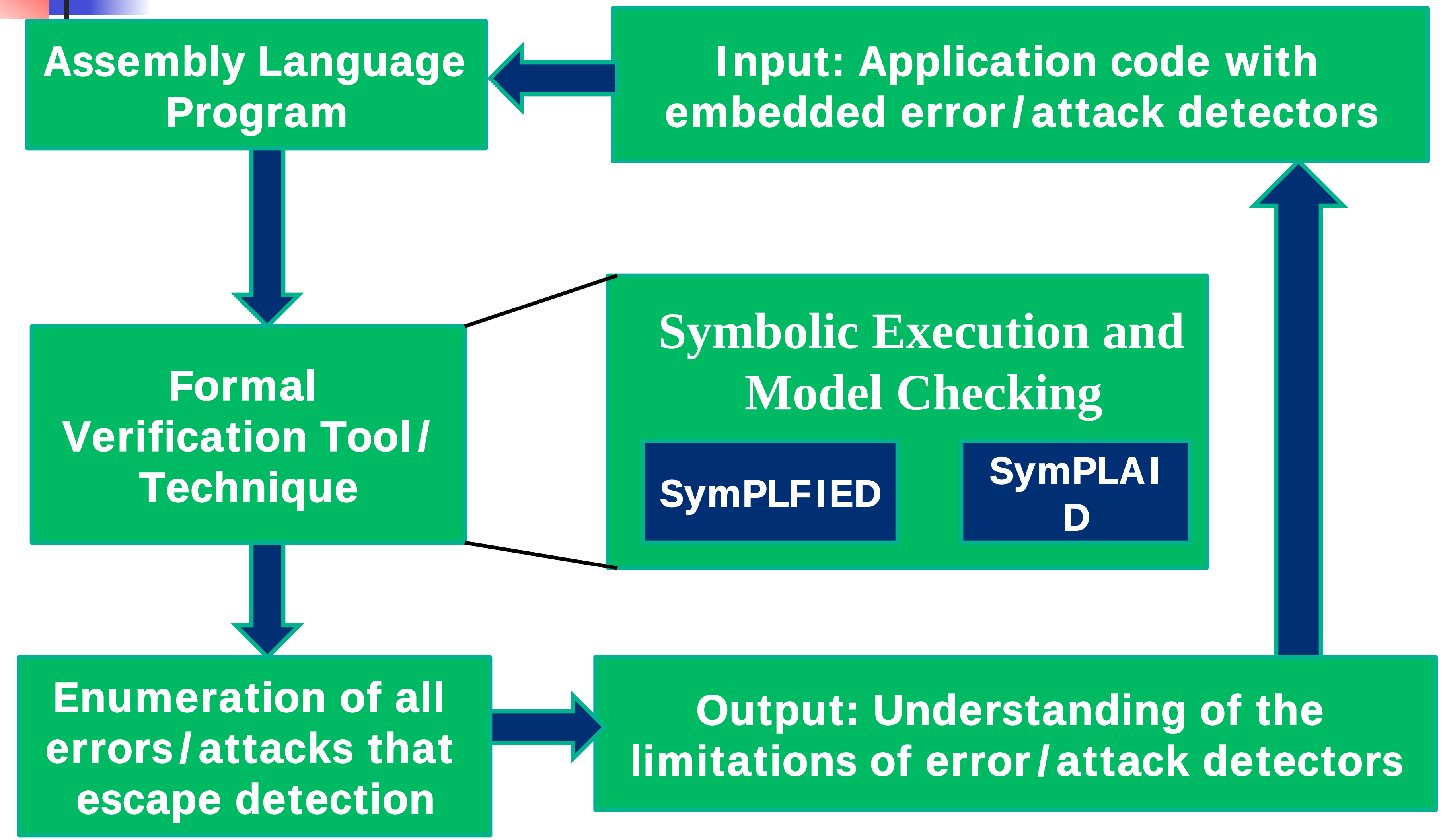


H/W Implementation - RSE



Validation Using Symbolic Execution and Model-Checking Framework

Formal Framework for Software and Detector Validation



SymPLFIED: (Symbolic Program Level Fault-Injection and Error Detection Framework)

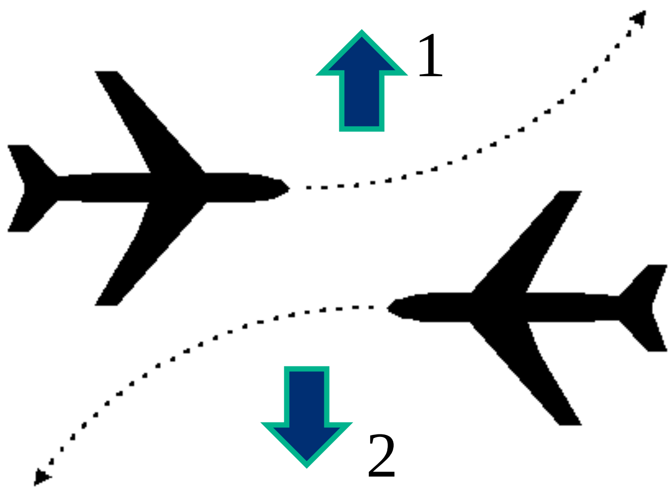
- **Goal:** evaluate the effects of runtime errors on programs with detectors
- Analyze programs directly in assembly language
- **Generic representation of error detectors**
  - Allows arbitrary error detectors to be specified in application
- **Fault Model:** Hardware (memory and processor) and (some) software errors

- **Comprehensive enumeration of undetected errors that lead to program failure**

ILLINOIS

SymPLFIED: Case Study

- **Tcas: Application Characteristics**
  - FAA mandated Aircraft collision avoidance system
  - Rigorously verified protocol and implementation
  - About 150 lines of C code = 1000 lines of assembly



Inputs: Positional parameters of other aircraft (and self)

Outputs:

0 – Unresolved

1 – Ascend

2 - Descend

Summary

- Derivation error and attack detectors
  - Using application-properties discovered through static and dynamic analysis
  - Detectors derived from backward slice of critical variables in the application
  - Derived detectors can be implemented in programmable hardware and software
- Future Directions
  - Controlled Diversity
  - Formal Techniques
  - Hardware Compilation